Evaluating the Applicability of Reliability Prediction Models between Different Software

Shin-ichi Sato
NTT DATA Corporation
3-3-3, Toyosu, Koto-ku, Tokyo, Japan, 135-6033
+81-3-5546-8384
satousnb@nttdata.co.jp

Akito Monden
Nara Institute of Science and Technology
8916-5, Takayama, Ikoma, Nara, Japan, 630-0101
+81-743-72-5312
akito-m@is.aist-nara.ac.jp

Ken-ichi Matsumoto
Nara Institute of Science and Technology
8916-5, Takayama, Ikoma, Nara, Japan, 630-0101
+81-743-72-5312
matumoto@is.aist-nara.ac.jp

Abstract

The prediction of fault-prone modules in a large software system is an important part in software evolution. Since prediction models in past studies have been constructed and used for individual systems, it has not been practically investigated whether a prediction model based on one system can also predict fault-prone modules accurately in other systems. Our expectation is that if we could build a model applicable to different systems, it would be extremely useful for software companies because they do not need to invest manpower and time for gathering data to construct a new model for every system.

In this study, we evaluated the applicability of prediction models between two software systems through two experiments. In the first experiment, a prediction model using 19 module metrics as predictor variables was constructed in each system and was applied to the opposite system mutually. The result showed predictors were too fit to the base data and could not accurately predict fault-prone modules in the different system. On the basis of this result, we focused on a set of predictors showing great effectiveness in every model; and, in consequent, we identified two metrics (Lines of Code and Maximum Nesting Level) as commonly effective predictors in all the models. In the second experiment, by constructing prediction models using only these two metrics, prediction performance were dramatically improved. This result suggests that the commonly effective model applicable to more than two systems can be constructed by focusing on commonly effective predictors.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification – *reliability, statistical method.*

General Terms

Reliability

Keywords

fault-prone module, metrics, prediction, software measurement, software reliability

1. Introduction

It is nothing new to say that quality is an important attribute of software systems. If a failure occurs while software is doing critical tasks, it may cause serious damage not only to the company who relies on the software but also to the company who is responsible in maintaining the software. Therefore, keeping up the reliability of software throughout its long-term evolution is a crucial work for the company who maintains it. However, testing and reviewing all the modules in large-scale software is very expensive. In order to lessen the cost of maintenance, a method for selecting modules that should be tested and/or reviewed is needed. One way to achieve this is to identify fault-prone modules in advance by using prediction models so that we can focus on the modules that need intensive testing and hence detect faults more efficiently.

In order to detect fault-prone modules, many studies have proposed prediction models using various analysis methods based on software product metrics. Selby and Porter proposed a decision tree approach to classify low-quality modules based on information theory and Huffman Coding [9]. Takahashi and his colleagues improved the tree construction algorithm by using AIC (Akaike's Information Criteria) [12]. Ohlsson and Alberg identified fault-prone modules in telephone switching software using multiple regression analysis [7]. They evaluated the accuracy of several models using Alberg Diagram. Pighin and Zamolo classified fault-prone modules using discriminant analysis [8]. Takabayashi and his colleagues predicted fault-prone modules using a neural network model [11]. Khoshgoftaar and Allen analyzed effective metrics to classify fault-prone modules using logistic regression analysis [4].

Most of these results showed that there is a close relationship between module metrics and its quality, and that all of these prediction models could predict fault-prone modules at high accuracy. However, metrics and analysis methods used in these

Table 1 Profile of Target Software

Name	S1	S2
Platform	Mainframe	Mainframe
Language	Two Hardware Specific Languages (Lang A and Lang B)	COBOL
Size	3,858 Modules (1.8MSLOC) Lang A: 1,995 Modules Lang B: 1,863 Modules	500 Modules (157KSLOC)
Fault data	In the 7 years after the systems was released	During unit testing and integration testing

studies are various and different case by case. Furthermore, most of results of these studies were validated on only a single software system. This indicates that a prediction model based on one software system is not necessarily effective for other software systems to predict fault-prone modules. Our expectation is that if we could build a model applicable to different systems, it would be extremely useful for software companies because they do not need to invest manpower and time for gathering data to construct a new model for every system.

In this study, we conducted an experimental research to evaluate the applicability of fault-prone module prediction model between different software. We conducted two experiments using data collected from two industrial software systems. As a result of the experiments, we identified two metrics that are commonly effective to predict fault-prone modules for different software and evaluated the applicability of prediction models between different software.

In the rest of this paper, we explain our experiments and show their results in Section 2. In Section 3, we have a discussion about the results in detail, and finally summarize the paper in Section 4.

2. Our Approach

We conducted an experiment to evaluate the applicability of prediction models using data collected from two industrial software systems that were developed in NTT DATA. The profile of them is shown in Table 1.

Software 1(referred as S1) is application software for some public institute. The first version was released about more than 20 years ago. Then it has been updated and changed many times to fix faults and to add new functionality till today. Besides of such occasional modifications, it has been renewed a few times due to changing its platform. At that time, some modules were abandoned and rewritten often in a different programming language because they became too old and complex by modifications over a long term. So, each module of today's version is written in either of two languages. Both of these are specific to its platform (mainframe) and it is not known in public. For the sake of convenience, we call them "Lang A" and "Lang B", respectively. Lang A has been mainly used in the early days and its syntax looks like that of PL/I. Lang B is an expansion of COBOL and used in later development. Many developers prefer Lang B to Lang A because it is newer and is more familiar to them than Lang A. The characteristics of these two languages are so different that we regard each module written in each language as a different software system.

Software 2(S2) is utility software. Each module of S2 is written in COBOL. Recently, when an old software system in our company needed to change its platform on account of a customer, S2 was developed for converting a database from old platform to new one. So S2 is operated on mainframe, but it is not legacy software.

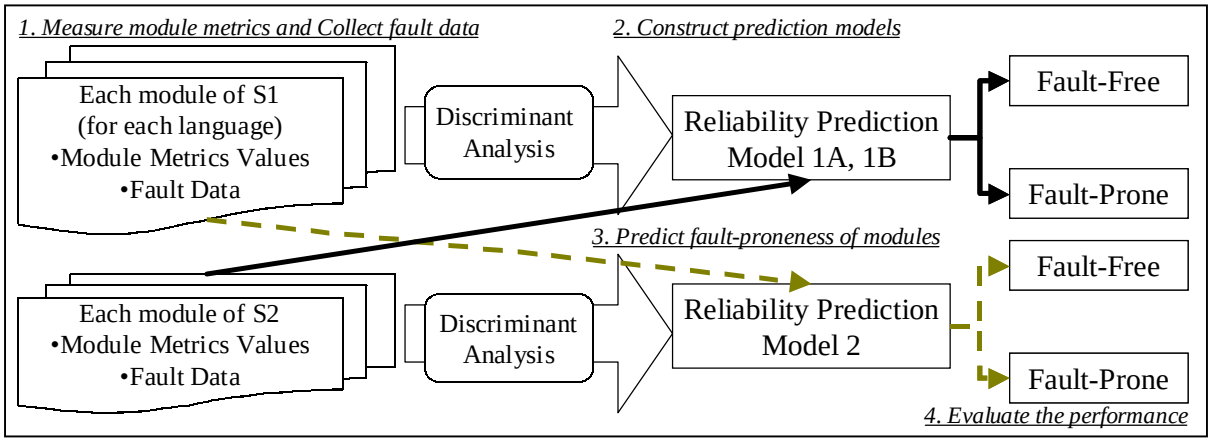

Figure 1 Overview of Experiment 1

Table 2 List of Module Metrics

Category	Metric Name	Abbreviated Name
Size	SLOC(Source Lines Of Code)	SLOC
Condition	Maximum Cyclomatic Number	MAX_CYC
	Cyclomatic Number / SLOC	N_CYCLO
	Jump Statements / SLOC	N_JUMP
	Loop Statements / SLOC	N_LOOP
Nest	Maximum Nesting Level	MAXNST
	Total Nesting Level / SLOC	N_NSTSUM
Data Structure	Number of Declared Variables / SLOC	N_VDESUM
	Number of Variable Uses / SLOC	N_VREFS
	Number of Variable Defines / SLOC	N_VASSUM
Understandability	(1-Comment Lines) / SLOC	N_COMM
Module Cohesion	Number of Procedure Calls / SLOC	N_INCSUM
Module Coupling	Number of External Module Calls / SLOC	N_EXCSUM
	Number of External Variable Defines / SLOC	N_EXASS
	Number of External Variable Uses / SLOC	N_EXREF
	Number of Macro References / SLOC	N_MACROS
	Number of Parameters	PARAMS

The size of each system is very different. S1 is considerably larger than S2. Also, the timings of collecting fault data are different: in maintenance phase for S1 and in testing phase for S2.

That is to say, S1 and S2 have quite different characteristics in their language, size, functionality and detected faults; and furthermore, modules in S1 have two different characteristics in written languages.

2.1 Experiment 1

2.1.1 Procedure

The procedure of the first experiment is shown in Figure 1. Each step of this procedure is explained in the followings.

1. Measure module metrics and collect fault data

The list of module metrics used in our study is shown in Table 2. All of these metrics are often used in other studies to evaluate module's complexity from various points of view, which include size metrics, software science metrics [2], McCabe's cyclomatic complexity metrics [5], and module cohesion metrics [6]. We measured these metrics in each module using a tool developed by ourselves. Since some of them were highly correlated with SLOC, so we normalized them by dividing by SLOC. Fault data were manually collected from failure reports.

Table 3 Two-class Classification Performance Matrix

		Prediction Results	
		Fault-Free	Fault-Prone
Actual Results	Fault-Free	n_{11}	n_{12}
	Fault-Prone	n_{21}	n_{22}

2. Construct prediction models for each software system.

As a prediction model, we used linear discriminant analysis whose dependent variable is a category variable (its value is "Fault-Prone" or "Fault-Free") and independent variables are module metrics. We constructed prediction models for each of S1 and S2. As mentioned in the previous section, each module of S1 has different characteristics in its written language, so we divided modules of S1 into two sets by its language and constructed prediction models for each set. That is, three prediction models were constructed: two models (Model 1A, 1B) from S1 and one model (Model 2) from S2.

3. Predict fault-proneness of modules using constructed models

We applied each module of one software into the prediction model constructed from another software to evaluate the effectiveness of the prediction model between different software: modules of S1 were applied into Model 2 and modules of S2 applied into Model 1A and 1B.

4. Evaluate the prediction performance.

The predicted modules can be categorized into four segments shown in Table 3. Each n_{ij} means the number of modules mapped into the corresponding segment. We used the following measures derived from Table 3 to evaluate the prediction performance. These measures were used in previous studies [1].

- Accuracy

This is a percentage of correctly predicted modules about both of fault-free modules and fault-prone modules. It is defined as:

$$\left(\frac{n_{11} + n_{22}}{n_{11} + n_{12} + n_{21} + n_{22}} \right) \times 100$$

- Correctness

This is a percentage of modules that were predicted as "Fault-Prone" and actually included one or more faults. If this value is low, that means the model is predicting more modules as "Fault-Prone" but they really are not faulty. It is defined as:

$$\left(\frac{n_{22}}{n_{12}+n_{22}}\right)\times 100$$

- Completeness

This is a percentage of modules including one or more faults and was predicted as "Fault-Prone". If this value is low, this means that more faulty modules are mis-predicted as "Fault-Free". It is defined as:

$$\left(\frac{n_{22}}{n_{21}+n_{22}}\right)\times 100$$

2.1.2 Result and Consideration

The result of experiment 1 is shown in Table 4. The upper table is the result of applying modules of S1 into Model B and the lower is the result of applying modules of S2 into Model 1A and 1B.

As a whole, Completeness showed pretty good results, however, Accuracy and Correctness were pretty bad. Especially, in the result of S2 (the lower table of Table 4), Completeness showed 100%. In fact, this is because all modules were predicted as "Fault-Prone". Similarly, almost all modules were predicted as "Fault-Prone" in S1. As a result, despite of the goodness of Completeness, the model could not predict fault-prone modules effectively in both software systems.

Table 4 Result of Experiment 1

S1	Modules written in Lang A	Modules written in Lang B
Accuracy	16.30%	56.29%
Correctness	13.60%	10.22%
Completeness	92.45%	71.97%

S2	Model 1A	Model 1B
Accuracy	16.20%	16.20%
Correctness	16.20%	16.20%
Completeness	100%	100%

2.2 Experiment 2

2.2.1 Motivation

From the prediction model constructed using discriminant analysis, we can not only predict fault-prone modules but also examine how effective each predictor variable (module metric) in the discriminant function is to predicting a fault-prone module by examining within-groups correlations of each variable with the canonical variable [10].

Table 5 shows the within-groups correlations of each predictor variable in each model. The name of each metric is the same as Table 2. Only top 5 of predictor variables sorted by correlation value are listed. In every model, the most effective metric is SLOC. For other metrics, the rank of effectiveness is different. However, in each model, there is the borderline (double line in the table) at which the correlation value remarkably decreases: 0.5274 to 0.3852 in Model 1A, 0.6265 to 0.3570 in Model 1B and 0.6559 to 0.4956 in Model 2.

We took metrics ranked above the border as the effective metrics in each model and investigated in detail. Then, we could find that MAXNST was the second metric that is effective in all models. From this result, we assumed that these two metrics (SLOC and MAXNST) would be commonly effective metrics independent of characteristics of software and that other metrics would be effective but specific to each software system.

2.2.2 Procedure

We conducted the second experiment similar to the procedure of experiment 1 except using only two metrics mentioned in previous section as the predictor variables in model construction. Other components of the experiment such as model construction method and measures to evaluate the prediction result are the same as experiment 1.

2.2.3 Result and Consideration

The prediction results of experiment 2 are shown in Table 6. Accuracy was drastically improved in all models, whereas, Completeness and Correctness were worse instead. However, considering the lower ratio of actual fault-prone modules in all modules, these results seem to be better than results of experiment 1. We will consider these results in detail in next section.

3. Discussions

In order to investigate whether the two metrics listed in experiment 2 are commonly effective or not, we evaluated results of both experiments by another measure: module-order model [3]. In this measure, predicted modules are listed in decreasing order of their fault-proneness so that the fault-prone modules are

Table 5 Within-groups correlations of each predictor variable with the canonical variable

Model 1A	
SLOC	0.6928
N_VEXASS	0.6102
N_VREFS	0.5582
MAXNST	0.5274
N_CYCLO	0.3852

Model 1B	
SLOC	0.8689
MAXNST	0.6265
M	0.357
N_VEXASS	0.3144
N_VASSU	0.2232

Model 2	
SLOC	0.7816
N_VASSU	0.7479
MAXNST	0.6559
N_VREFS	0.4956
N_COMM	0.4531

Table 6 Results of Experiment 2

S1	Modules written in Lang A	Modules written in Lang B
Accuracy	83.91%	67.34%
Correctness	24.11%	22.40%
Completeness	66.67%	53.58%

S2	Model 1A	Model 1B
Accuracy	76.60%	84.20%
Correctness	36.76%	51.56%
Completeness	61.73%	40.74%

emerged at the beginning. When we extract some fixed number of modules (e.g. only 10% of whole modules) from the beginning of the list, we could evaluate the performance of a prediction model by the number of faulty modules included in the extracted modules.

The results by module-order model are shown in Figure 2 and Figure 3. They are illustrated by diagrams similar to Alberg Diagram [7]. Their x-axis mean the ratio of modules placed in decreasing order of the fault-proneness and y-axis mean accumulated ratio of extracted modules with faults. Thin lines are plotted results of experiment 1 and bold lines are plotted result of experiment 2. Straight lines with one fold mean the ideal results (it would be shaped if all fault-prone modules are exactly listed in the higher rank of the result). So this means that the more convex to upper left the shape is, the better the prediction performance is.

From Figure 2, we can find that the prediction performance of module listed in upper rank is better for the result of experiment 2 than the one of experiment 1. And in Figure 3, the difference is very clear. Prediction performances were dramatically improved in experiment 2.

That is to say, prediction performance was improved by using only commonly effective metrics (SLOC and MAXNST). We guess the reason is that prediction models constructed using all the metrics fit to the base data too much, so that they cannot predict fault-prone modules in other software.

Considering selected metrics, SLOC is the most effective metrics in all the prediction models. This has been often said before, however, it is not enough. In this research, we could make it clear that there is another metric (Maximum Nesting Level), which is commonly effective to predict fault-prone modules.

4. Conclusion

In this research, we experimentally evaluated the applicability of reliability prediction models between different software systems. Prediction models based on many metrics were too fit to the base data so that prediction performance for other software was very bad. By using only commonly effective metrics, prediction performance was drastically improved.

We believe that this result is very useful to predict fault-prone modules in newly developed software by constructing the prediction model based on commonly effective metrics. By conducting similar studies using more metrics, we will be able to identify more metrics that are commonly effective to fault-proneness.

References

[1] Basili R. V., Condon E. S., Emam E. K., Hendrick, B. R. and Melo, W., “Characterizing and Modeling the Cost of Rework in a Library of Reusable Software Components”, In Proceedings on the 19th International Conference on Software Engineering, pp. 282-291, 1997.

[2] Halstead, M. H., “Elements of Software Science”, ELSEVIER COMPUTER SCIENCE LIBRARY, 1977.

[3] Khoshgoftaar, T. M. and Allen, E. B., “A Comparative Study of Ordering and Classification of Fault-Prone Software Modules”, Empirical Software Engineering, Vol. 4, 159-186, 1999.

[4] Khoshgoftaar, T. M. and Allen, E. B., “Logistic Regression Modeling of Software Quality, International Journal of Reliability”, Quality and Safety Engineering, Vol. 6, No. 4, pp. 303-317, 1999.

[5] McCabe, T. J., 1976. A Complexity Measure, IEEE Transactions on Software Engineering, Vol. SE-2, 308-320.

[6] Marciniak. J. 1994. Encyclopedia of Software Engineering, John Wiley & Sons Inc., New York.

[7] Ohlsson, N. and Alberg, H., “Predicting Fault-Prone Software Modules in Telephone Switches, IEEE Transactions on Software Engineering”, Vol. 22, No. 12, pp. 886-894, 1996.

[8] Pighin, M. and Zamolo, R., “A Predictive Metric Based on Discriminant Statistical Analysis”, In Proceedings on the 19th International Conference on Software Engineering, pp. 262-270, 1997.

[9] Selby, R. W. and Porter, A. A., “Learning from Examples: Generation and Evaluation of Decision Trees for Software Resource Analysis”, IEEE Transactions on Software Engineering, Vol. 14, No. 12, pp. 1743-1757, 1988.

[10] SPSS Inc., “SPSS Base 10.0 Application Guide”, 1999.

[11] Takabayashi, S., Monden, A., Sato, S., Matsumoto, K. and Torii, K., ”The detection of fault-prone program using a neural network”, In Proceedings on the International Symposium on Future Software Technology '99.

[12] Takahashi, R., Muraoka, Y. and Nakamura, Y. “Building Software Quality Classification Trees: Approach, Experimentation, Evaluation”, In Proceedings on the 8th International Symposium on Software Reliability Engineering (ISSRE '97), pp. 222-233, 1997.

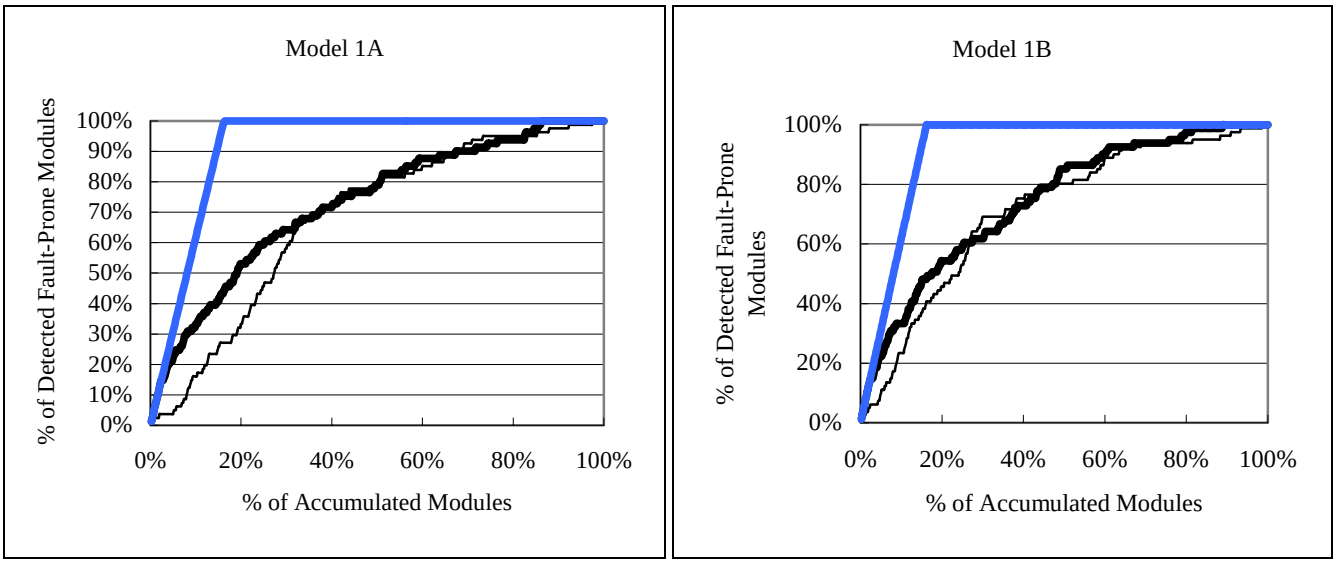

Figure 2 results by module-order model (Model 1A and 1B)

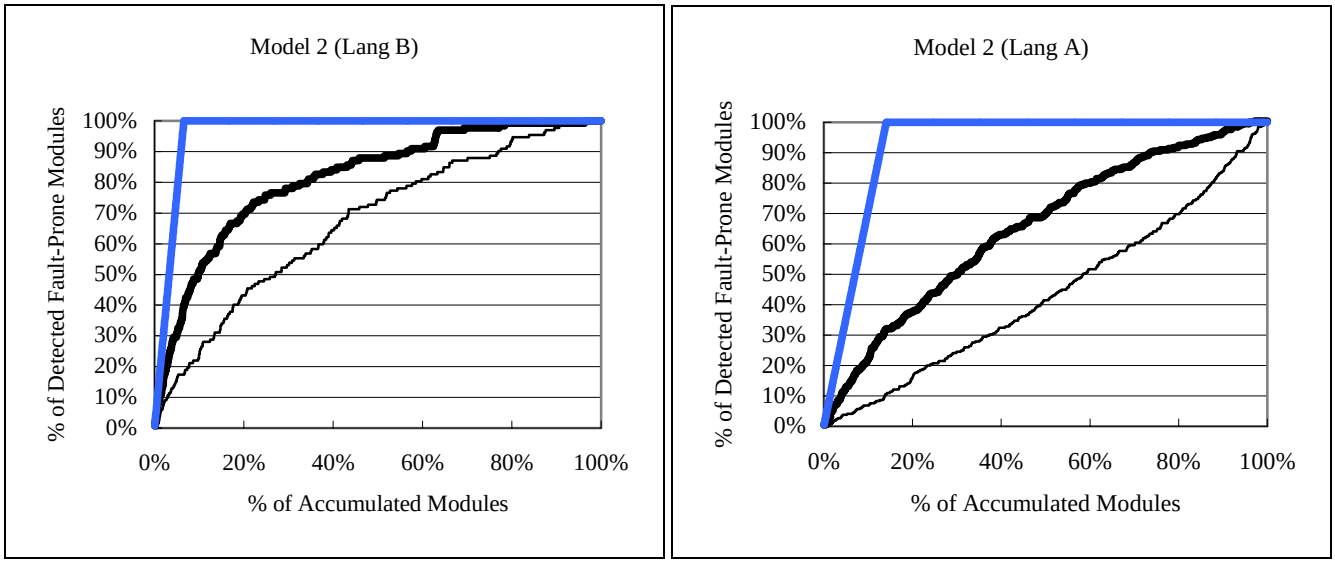

Figure 3 Results by module-order model (Model 2 for Lang A and B)